Universität Bremen
FB 3 – Informatik
Prof. Dr. Rainer Koschke
TutorIn: Euer/Eure TutorIn

Software–Projekt 2 2012/13

VAK 03-BA-901.02

Testplan der Gruppe Power Rangers

Simon Auchter	sauchter@tzi.de	1234567
Dennis Brunkhorst	denbrunk@tzi.de	1234567
Marcel Luttermann	marcel30@tzi.de	1234567
Mohammad Mehdi Salem Naraghi	mehdi.salem@uni-bremen.de	1234567
Ingo Schnell	ingos@uni-bremen.de	1234567
Himmet Yumusak	himmet.yumusak@uni-bremen.de	1234567

Abgabe: 23. Dezember 2012 — Version 1.0

Inhaltsverzeichnis

Version und Änderungsgeschichte

Die aktuelle Versionsnummer des Dokumentes sollte eindeutig und gut zu identifizieren sein, hier und auf dem Titelblatt.

Version	Datum	Änderungen
1.0	23.12.2012	Abgabefertiges Dokument

1 Einführung

1.1 Zweck

Dieser Testplan bietet einen ersten Überblick über die geplanten Testverfahren, mit denen wir unsere Software ausführlich auf Funktionalität testen wollen. Wir werden den Testplan während der Implementierungsphase ständig benutzten, um den geschriebenen Code zu auf Funktionalität zu testen.

1.2 Umfang

1.3 Beziehungen zu anderen Dokumenten

Der Testplan wird mehrere Referenzen zur Architekturbeschreibung aufweisen, da für die Testspezifikationen auf festgelegten Module und Komponenten eingegangen wird. Dazu werden auch bereits erstellte Diagramme des Entwurfs wiederverwendet. Außerdem wird die Modulsicht helfen die JUnit Tests zu erstellen. Einen weiteren Einfluss auf dieses Dokument nimmt die Anforderungsspezifikation, wo die kompletten Systemeigenschaften und Systemattribute bereits beschrieben wurden.

1.4 Aufbau der Testbezeichner

1.5 Dokumentation der Testergebnisse

Zu den erstellten Testfällen wird jeweils ein kurzes Protokoll anfertigt, welches den Ablauf des Tests zusammenfasst und mögliche Komplikationen während der Durchführung beinhaltet. Anschließend wird das Resultat des Test bestimmt. Gefundene Fehler werden bestmöglich beschrieben, in das Protokoll eingefügt und dann zeitnah bearbeitet.

1.6 Definitionen und Akronyme

Begriff	Definition
Android	Betriebssystem für mobile Geräte (hauptsächlich Smartphones und Tablets).
Anwendungsserver	Server in einem Netzwerk, auf dem Anwendungsprogramme ausgeführt werden.
App (Applikation)	Im Kontext dieses Dokuments ist mit *App* eine Android Applikation gemeint, d.h. ein Android Programm dass auf dem Handy ausgeführt wird.
Architektur	Hier: beschreibt die Konfiguration der einzelnen Computersysteme und ihre Beziehung zueinander.
Betriebssystem	In unserem Kontext eine Schnittstelle zwischen Hardwarekomponenten und der Anwendungssoftware des Benutzers.
Client	hier: Gerät, dass eine Dienstleistung von einem Server anfordert.
Cluster	Hier: ein Rechnerverbund, dessen einzelne Computer gemeinsam ein hohes Rechenaufkommen bewältigen.
Datenbank	System zur elektronischen Datenverwaltung.
Datenhaltungsschicht	Die Schicht in der Architektur, die für die Datenhaltung zuständig ist.
Features	Zusätzliche Funktionen.
Framework	Stellt einen Programmiergerüst zur Entwicklung von Software bereit.
Frontend	Hier fehlt noch die Definition
GUI	Graphical User Interface. Benutzerschnittstelle mit deren Hilfe der Benutzer mit der Software interagiert.
Hardware	In unserem Dokument eine mechanisch und elekronische Ausrüstung eines Computersystems.
Implementierungssprache	Genutzte Programmiersprache um das Projekt zu entwickeln.
Java	Weitverbreitete objektorientierte Programmiersprache, die vor Allem wegen ihrer Plattformunabhängigkeit beliebt ist.
JBoss	Freie Software, die eine Implementierung eines Anwendungsservers, bereitstellt.
Logikschicht	Die Schicht in der Architektur, die für die Logik des System zuständig ist.
Modularisierung	Modularität ist das Prinzip der Aufteilung eines Ganzen in Teile, welche als Module bezeichnet werden. Bei geeigneter Form lassen sie sich zusammenfügen oder interagieren über Schnittstellen miteinander.
Netzwerkprotokoll	Protokoll für den Austausch von Daten zwischen verbundenen Computern.
OpenSource	Kostenlos verwendbare Software.
Persistenz	Fähigkeit zur Bereitstellung von Daten oder Objekten über einen längeren Zeitraum.

Begriff	Definition
Plattform	Ebene innerhalb eines Computersystems, welche für die Ausführung und Erstellung von Programmen benötigt wird.
Plattform-unabhängig	Software oder Dateien sind plattformunabhängig, wenn sie nicht von bestimmten Betriebssystemen abhängig sind und auf unterschiedlichen Plattformen verwendet werden können.
Revision	Synonym für den Begriff Version.
Server	Bezeichnet hier einen Computer in einem Netzwerk, auf dem Software läuft, die den Clients Zugang zu Dienstleistungen ermöglicht.
Software	Ausführbares Programm, eventuell mit zugehörigen Daten.
SQLite	SQLite ist eine Programmbibliothek, die ein relationales Datenbanksystem bereitstellt.
TCP/IP	Netzwerkprotokoll, welches eine große Bedeutung für das Internet hat.
Tool	Computer Werkzeug, kleine Anwendungssoftware oder Dienstprogramm.
UML	Grafische Modellierungssprache zur Spezifikation, Konstruktion und Dokumentation von Software-Teilen.
XML-Datei	Extensible Markup Language - Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten.

1.7 Referenzen

1. Android
http://www.android.com/
2. Dreischichtige Architektur
http://de.wikipedia.org/wiki/Schichtenarchitektur#Drei-Schichten-Architektur
3. Eclipse
http://www.eclipse.org/
4. Hibernate
http://www.hibernate.org/
5. Java
https://www.java.com/de/
6. JavaDoc
http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

7. JBoss
http://www.jboss.org/jbossas/
8. Latex, MikTex
http://miktex.org/
9. Lernsysteme (3-Kasten-Prinzip, Leitner-Prinzip)
(https://de.wikipedia.org/wiki/Lernkartei)
10. UML (Unified Modeling Language)
http://www.uml.org/
11. Visual Paradigm
http://www.visual-paradigm.com/
12. Zweischichtige Architektur
http://de.wikipedia.org/wiki/Schichtenarchitektur#Zwei-Schichten-Architektur

2 Systemüberblick

Das System ist in zwei Pakete aufgeteilt, die als `Client` und `Server` bezeichnet werden. Der Testplan bezieht sich auf die konzeptionelle Sicht der Architekturbeschreibung und verwendet die dort ermittelten Komponenten innerhalb der Pakete. Diese Komponenten haben untereinander starke Abhängigkeiten und liefern eine gute Grundlage für die Erstellung von den Komponententests, da diese Abhängigkeiten beibehalten werden können. Hierbei ist besonders wichtig, dass die Zusammenarbeit der `Logic` Komponenten der Pakete mit den übrigen Komponenten des Pakets fehlerfrei funktioniert.

2.1 Module der Anwendungsschicht und deren Funktionen

Hier solltet ihr dann die einzelnen Module aus Sicht des Tests weiter verfeinern. Auch wäre noch eine Grafik angebracht, die die Abhängigkeiten der Subsysteme/Module zeigt, sowie eine kurze Beschreibung dazu.

3 Merkmale

Es werden die Merkmale bestimmt, die ausführlich getestet werden müssen, um das Ziel der höchstmöglichen Benutzerfreundlichkeit zu erreichen. Dabei wird besonders darauf geachtet, dass vorgegebene Mindestanforderungen erfüllt sind und diese auch zufriedenstellend funktionieren.

3.1 Zu testende Merkmale

Das wichtigste Gesamtziel ist es, den Benutzern der Applikation eine möglichst intuitive und problemlose Interaktion mit der Applikation ermöglichen. Die Funktionen, die dies sicherstellen, werden wir vorrangig auf Funktionalität testen.

3.1.1 Funktionale Anforderungen

Es muss sichergestellt sein, dass der Benutzer einen reibungslosen Ablauf während des Lernens erfährt. Hierbei müssen vor allem die Funktionen des Karteikastens getestet werden. Dabei handelt es sich spezifischer um die korrekte Darstellung der Karteikarten und das anschließende Anzeigen der Definitionen. Des Weiteren muss das Bewerten der Karteikarten korrekt funktionieren und die Einstufung des Wissenstand für eine Karte. Hierbei muss sichergestellt werden, dass der Fortschritt auch bei Unterbrechen des Lernens korrekt gespeichert wird.

Eine weitere wichtige Anforderung ist, dass die Kommunikation zwischen Client und Server korrekt funktioniert und daraus resultierend alle Daten richtig versendet werden. Da die Applikation im Auslieferungszustand noch keine Karteikarten enthält, muss der erstmalige Abruf der Karten reibungslos funktionieren, um einen einfachen Einstieg in den Lernmodus zu garantieren.

Zusätzlich wird versucht, dass Funktionen, die für die Mindestanforderungen erforderlich sind, ausreichend getestet werden, um so sicherzustellen, dass die Software den Anforderungen entspricht.

3.2 Nicht zu testende Merkmale

Merkmale und Funktionen die nicht getestet werden sind vorrangig triviale Methoden, wie zum Beispiel jegliche `get` und `set` Methoden, hierbei gibt es natürlich Ausnahmen, die auch in der Modulsicht (*Javadoc*) gekennzeichnet sind. Des Weiteren wird sich bei den externen Bibliotheken (*JBoss, JSON, Hibernate*), darauf verlassen, dass durch eine gute Dokumentation und häufiger Verwendung in anderen Bereichen, die Funktionalität gewährleistet ist. Diese Bibliotheken werden jedoch passiv getestet, da wir ja besonderen Wert auf die Kommunikation legen. Desweiteren wird auch keine der von *Android* zur Verfügung gestellten Klassen getestet.

4 Abnahme- und Testendkriterien

Das System wird anhand von mehrere Ergebnisfaktoren bewertet und anschließend bewertet. Diese Bewertung ist ausschlaggebend dafür, wann wir das Testen beenden und für erfolgreich erklären. Bei Fehlschlagen der einzelnen Kriterien muss je nach Wertung des Fehlers schnellstmöglich eine Lösung gefunden werden.

Testabdeckung

Als Testabdeckung streben wir einen wert von 95 Prozent an, da wir sicherstellen wollen, dass vor allem die Mindestanforderungen alle abgedeckt sind.

Ergebnisse der Testfälle:

Es soll eine hohe Erfolgsquote in den Testfällen angestrebt werden. Wenn Testfälle fehlschlagen sollten müssen diese für eine Abnahme behoben werden. Hierbei ordnen wir die Art der Fehler nach einer Bewertung, welche in der folgenden Tabelle erläutert sind. Wir haben festgelegt, dass maximal eine zusammengefasste Einstufung von 5 erlaubt sind.

Abgabetermin:

Da der Auslieferungstermin für den 17.02.2013 festgelegt ist, müssen zu diesem Zeitpunkt zwingend die höchstmögliche Anzahl von Fehlern ausgemerzt sein, das ideal Ergebnis wäre eine fehlerfreie Software.

Fehlerwertung:

Wir haben die Auswirkung eines Fehler näher spezifiziert um diese nach Priorität ordnen zu können. Das ermöglicht es uns, gefundene Fehler effektiv zu bearbeiten.

Fehlerart	Beschreibung	Einstufung
Leicht	Fehler werden als leicht eingestuft, wenn sie nur geringe Auswirkungen auf den Programmfluss haben, es wäre möglich, ein lauffähiges Programm abzuliefern, welche Fehler dieser Art beinhaltet.	1
Mittel	Mittlere Einstufung erhalten Fehler, die bereits Auswirkungen auf den Programmfluss haben, jedoch sind die grundlegenden Funktionen noch funktionsfähig. Eine Auslieferung mit derartigen Fehlern sollte vermieden werden.	4
Schwer	Fehler, die schweren Einfluss auf die Funktionsfähigkeit der Software haben. Wenn elementare Funktionen der Software gestört werden können und das Risiko besteht, dass sie nicht mehr korrekt arbeiten.	8
Fatal	Dies betrifft Fehler, die eine Ausführung der Software unmöglich haben. Es sind grundlegende Funktionen betroffen und sollten schnellstmöglich behoben werden.	10

5 Vorgehensweise

5.1 Komponenten- und Integrationstest

Im Rahmen des Projects wurde eine Bottom-Up Herangehensweise als Strategie für die Integration der einzelnen Komponenten gewählt. Im Rahmen der Integrationstests testen wir zunächst die Komponenten des Servers und Clients unabhängig voneinander und testen anschließend die Funktionalität des Gesamtsystems.

Server

Das packet `server.data` deckt durch die Modul-Tests implizit das Packet `common.data` ab.

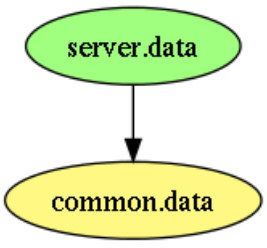

Abbildung 1: server.data

Analog gilt das für die Komponenten server.logic und common.logic.

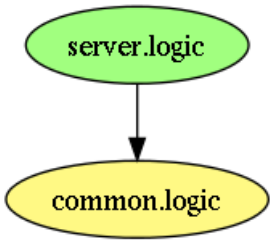

Abbildung 2: server.logic

Im nächsten Schritt fügen wir server.logic und server.data zusammen und testen diese gemeinsam.

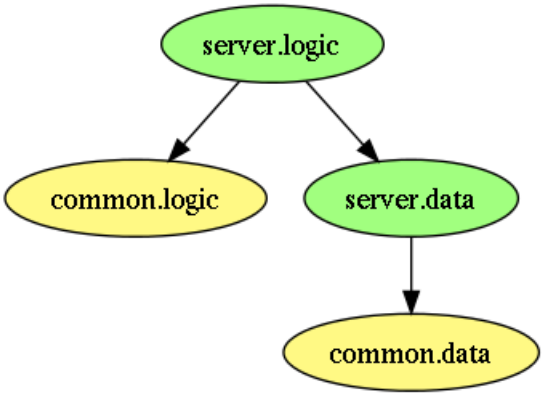

Abbildung 3: server.logic und server.data verbinden

Wir nehmen danach die Kommunikationsschnittstellen des Servers hinzu und testen diese gemeinsam. Damit wird der Server vollständig getestet.

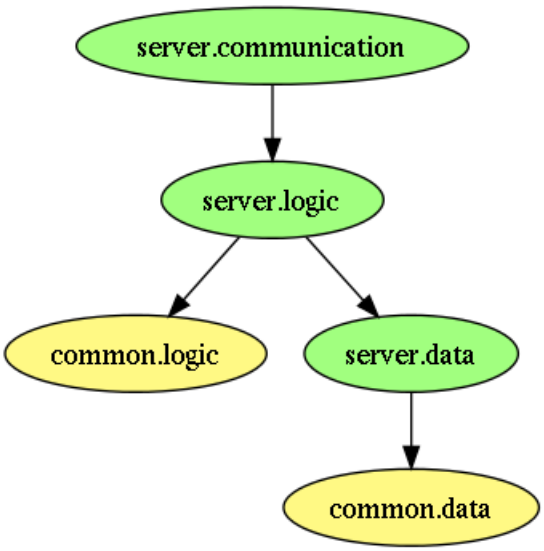

Abbildung 4: server.communication hinzunehmen

Client

Beim Client verfahren wir analog zum Server. Wir testen zunächst die Komponenten client.data und client.logic und damit implizt ihre Integration mit common.data und common.logic.

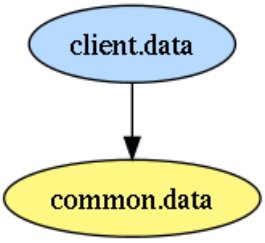

Abbildung 5: client.data

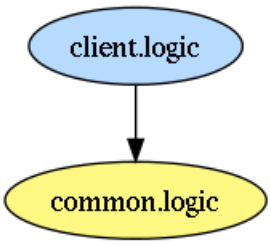

Abbildung 6: client.logic

Im nächsten Schritt integrieren wir client.data und client.logic und testen die resultierende Einheit.

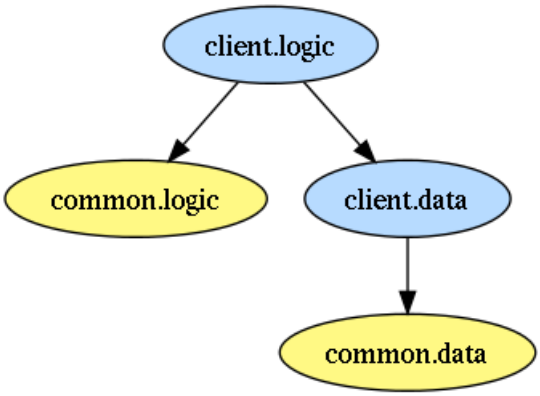

Abbildung 7: client.data und client.logic zusammenfügen

Als nächstes wird die Kommunikationsschnittstelle hinzugenommen und erneut getestet.

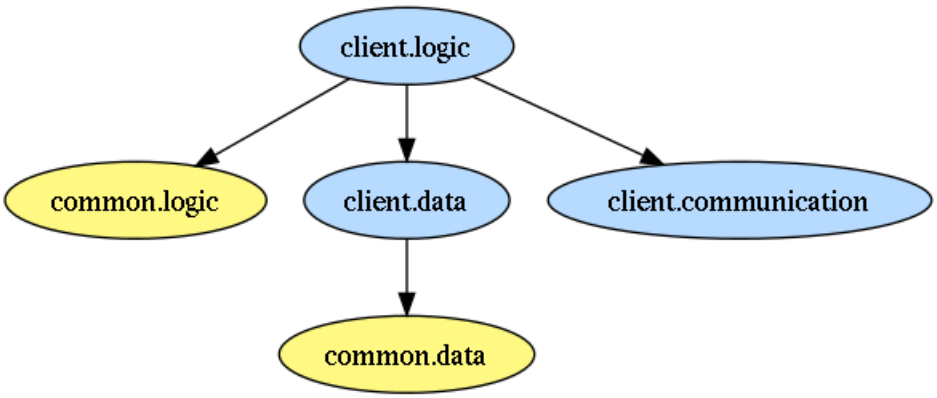

Abbildung 8: client.communication hinzunehmen

Das Userinterface schließt den Test des Clients ab, so dass davon ausgegangen werden kann dass der Client, für sich allein, den Anforderungen entspricht.

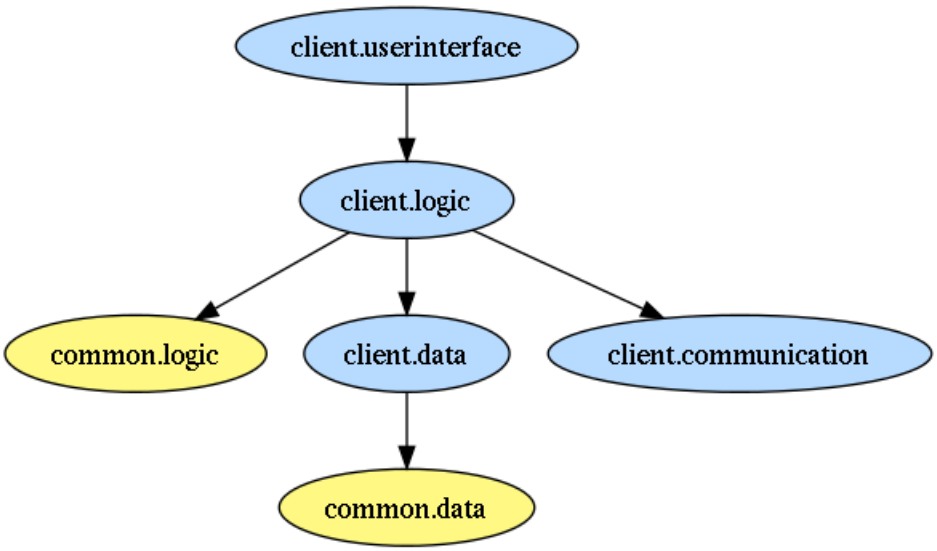

Abbildung 9: userinterface in die Tests miteinbinden

Interaktion von Client und Server

Im letzten Schritt testen wir das Gesamtsystem, indem wir Client und Server an den Kommunikationsschnittstellen zusammenfügen und erneut alles testen.

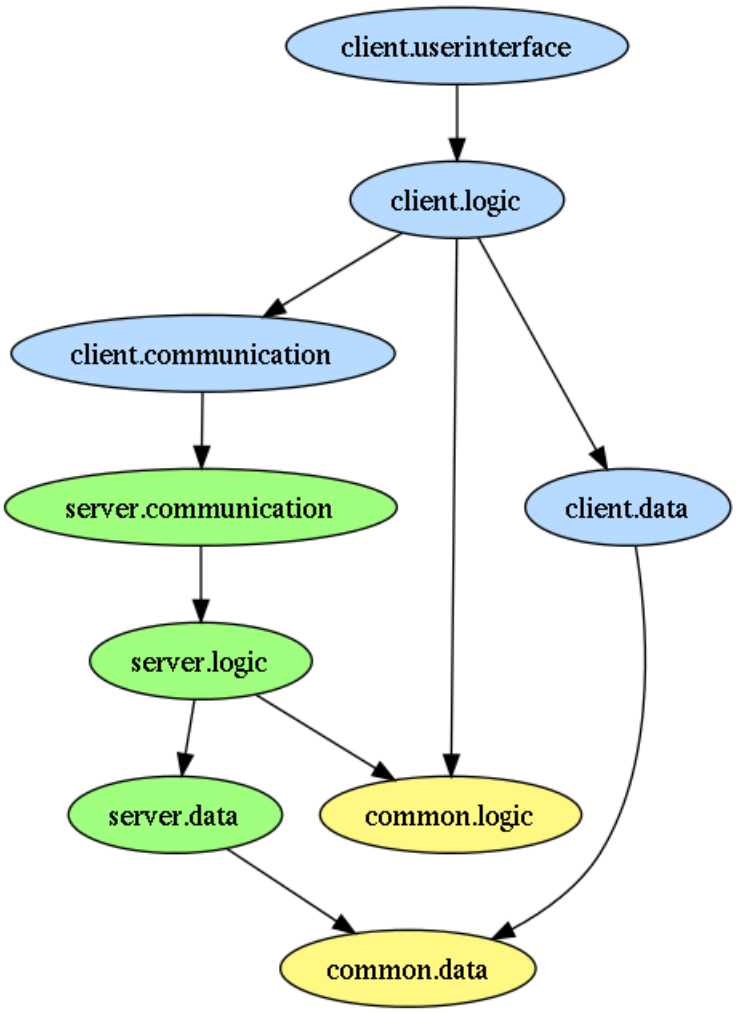

Abbildung 10: Server und Client gemeinsam testen

5.2 Funktionstest

Die Funktionstests werden von den Anwendungsfällen bestimmt, die wir in der Anforderungsspezifikation festgelegt haben. Jede in der Anforderungsspezfikation festgelegte Funktion muss durch ein Test gedeckt sein und einwandfrei funktioneren.

6 Aufhebung und Wiederaufnahme

Für Wiederaufnahme der Implementierung haben wir uns ein Maß von 20 festgelegt, welche aber keine schweren oder fatalen Fehler beinhalten darf. Diese werden sofort behoben.

7 Hardware- und Softwareanforderungen

Die Tests werden auf den verschiedenen Notebooks und Handys der Projektmitglieder durchgeführt und somit wird voraussichtlich keine weitere Hardware benötigt.

7.1 Hardware

Anforderungen an die Hardware sind ziemlich gering, da die Tests keine Ressourcenanforderungen besitzen. Zum Testen der Kommunikation zwischen Client und Server wird jedoch ein Server benötigt, der entweder durch einen virtuellen Server oder durch einen provisorischen Server auf einem der Notebooks repräsentiert wird. Des Weiteren sind Tests vorgesehen, die auf einem Android-Handy durchgeführt werden sollen. Dazu benötigen wir ein passendes Gerät.

7.2 Software

Größtenteils werden die Tests innerhalb von Eclipse durchgeführt, dabei handelt es sich vor allem um JUnit Tests. Des Weiteren werden Tests im Emulator von Android vorgenommen, um Funktionen der Applikation direkt zu testen.

8 Testfälle

Hier wird ist aufgelistet, was, wie, von wem getestet wird. Nach der Implementierung werden die Test anhand diesen Plans durchgeführt.

8.1 Komponententest

Klasse	Implementierer	Tester	Testart
server.data.Database	Simon Auchter	Mehdi Salem	Blackbox
client.data.Database	Mehdi Salem	Simon Auchter	Blackbox
client.logic.Cardbox	Marcel Luttermann	Ingo Schnell	Blackbox
common.logic.FileCard	Ingo Schnell	Marcel Luttermann	Blackbox
common.logic.Category	Dennis Brunkhorst	Himmet Yumusak	Blackbox
common.logic.Tag	Himmet Yumusak	Dennis Brunkhorst	Blackbox
common.logic.Tag	Simon Auchter	Mehdi Salem	Whitebox
common.logic.Category	Mehdi Salem	Simon Auchter	Whitebox
common.logic.FileCard	Marcel Luttermann	Ingo Schnell	Whitebox
client.logic.Cardbox	Ingo Schnell	Marcel Luttermann	Whitebox
client.data.Database	Dennis Brunkhorst	Himmet Yumusak	Whitebox
server.data.Database	Himmet Yumusak	Dennis Brunkhorst	Whitebox

Tabelle 1: Komponententests

8.2 Integrationstest

IT 0.0 Kategorie anlegen
Testobjekte: -client.userInterface -client.logic -client.data -common.logic **Eingabespezifikationen:** Kategorienamen,gegebenenfalls Liste von Kategorien **Ausgabespezifikationen:** Erfolgreiches anlegen einer Kategorie **Umgebungserfordernisse:** keine **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** keine

IT 0.1 Begriff hinzufügen
Testobjekte: -client.userInterface -client.logic -client.data -common.logic **Eingabespezifikationen:** Begriffname, Begriffdefinition, Liste von Kategorien **Ausgabespezifikationen:** Erfolgreiches anlegen eines Begriffs **Umgebungserfordernisse:** keine **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.0

IT 0.2 Karteikasten erstellen
Testobjekte: -client.userInterface -client.logic -client.data -common.logic **Eingabespezifikationen:** Karteikastenname, Liste von Begriffen, gewähltes Lernsystem **Ausgabespezifikationen:** Erfolgreiches anlegen eines Karteikastens **Umgebungserfordernisse:** keine **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.1

IT 0.3 Kategorie bearbeiten
Testobjekte: -client.userInterface -client.logic -client.data -common.logic **Eingabespezifikationen:** Zu bearbeitende Kategorie, Liste von Begriffen **Ausgabespezifikationen:** Erfolgreiches bearbeiten einer Kategorie **Umgebungserfordernisse:** keine **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.0

IT 0.4 Begriff bearbeiten
Testobjekte: -client.userInterface -client.logic -client.data -common.logic **Eingabespezifikationen:** Zu bearbeitender Begriff, Liste von Kategorien **Ausgabespezifikationen:** Erfolgreiches bearbeiten eines Begriffs **Umgebungserfordernisse:** keine **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.1

IT 0.5 Karteikasten bearbeiten
Testobjekte: -client.userInterface -client.logic -client.data -common.logic **Eingabespezifikationen:** Zu bearbeitender Karteikasten, Liste von Begriffen **Ausgabespezifikationen:** Erfolgreiches bearbeiten eines Karteikasten **Umgebungserfordernisse:** keine **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.1

IT 1.0 Kategorie anlegen
Testobjekte: -server.logic -server.data -common.logic **Eingabespezifikationen:** Kategorienamen,gegebenenfalls Liste von Kategorien **Ausgabespezifikationen:** Erfolgreiches anlegen einer Kategorie **Umgebungserfordernisse:** keine **Anforderungen:** Läuft in Lokaler Main Klasse, Server wird Simuliert **Abhängigkeiten:** keine

IT 1.1 Begriff hinzufügen
Testobjekte: -server.logic -server.data -common.logic **Eingabespezifikationen:** Begriffname, Begriffdefinition, Liste von Kategorien **Ausgabespezifikationen:** Erfolgreiches anlegen eines Begriffs **Umgebungserfordernisse:** keine **Anforderungen:** Läuft in Lokaler Main Klasse, Server wird Simuliert **Abhängigkeiten:** IT 1.0

IT 1.2
Kategorie bearbeiten
Testobjekte: -server.logic -server.data -common.logic **Eingabespezifikationen:** Zu bearbeitende Kategorie, Liste von Begriffen **Ausgabespezifikationen:** Erfolgreiches bearbeiten einer Kategorie **Umgebungserfordernisse:** keine **Anforderungen:** Läuft in Lokaler Main Klasse, Server wird Simuliert **Abhängigkeiten:** IT 1.0

IT 1.3
Begriff bearbeiten
Testobjekte: -server.logic -server.data -common.logic **Eingabespezifikationen:** Zu bearbeitender Begriff, Liste von Kategorien, bearbeitete Begriffdefinition **Ausgabespezifikationen:** Erfolgreiches bearbeiten eines Begriffs **Umgebungserfordernisse:** keine **Anforderungen:** Läuft in Lokaler Main Klasse, Server wird Simuliert **Abhängigkeiten:** IT 1.1

IT 2.0 Kategorie vom Server laden
Testobjekte: -client.userInterface -client.logic -client.data -client.communication -server.communication -server.logic -server.data -common.logic **Eingabespezifikationen:** Zum herunterladen gewählte Kategorie **Ausgabespezifikationen:** Erfolgreiches herunterladen der Kategorie **Umgebungserfordernisse:** Verbindung zum Internet, Server Online. **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.0, IT 0.1, IT 1.0 IT 1.1

IT 2.1 Begriff bewerten
Testobjekte: -client.userInterface -client.logic -client.data -client.communication -server.communication -server.logic -server.data -common.logic **Eingabespezifikationen:** Wählen der Bewertungspunktzahl(Sterne) beim Begriff **Ausgabespezifikationen:** Erfolgreiches bewerten des Begriffs **Umgebungserfordernisse:** Verbindung zum Internet, Server Online. **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.0, IT 0.1, IT 1.0 IT 1.1

IT 2.2
Begriff/Kategorie Updaten
Testobjekte: -client.userInterface -client.logic -client.data -client.communication -server.communication -server.logic -server.data -common.logic **Eingabespezifikationen:** Manuelles update, bzw. Automatisches **Ausgabespezifikationen:** Erfolgreiches Updaten des Begriff/Kategorie **Umgebungserfordernisse:** Verbindung zum Internet, Server Online. **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.0, IT 0.1, IT 1.0 IT 1.1

IT 2.3
Begriff vorschlagen
Testobjekte: -client.userInterface -client.logic -client.data -client.communication -server.communication -server.logic -server.data -common.logic **Eingabespezifikationen:** Gewählter Begriff **Ausgabespezifikationen:** Erfolgreiches vorschlagen des Begriffs **Umgebungserfordernisse:** Verbindung zum Internet, Server Online. **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.0, IT 0.1, IT 1.0 IT 1.1

IT 2.4 Kategorie vorschlagen
Testobjekte: -client.userInterface -client.logic -client.data -client.communication -server.communication -server.logic -server.data -common.logic **Eingabespezifikationen:** Gewählte Kategorie **Ausgabespezifikationen:** Erfolgreiches vorschlagen der Kategorie **Umgebungserfordernisse:** Verbindung zum Internet, Server Online. **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.0, IT 0.1, IT 1.0 IT 1.1

8.3 Funktionstest

Die Funktionstests werden anhand der Anwendungsfälle der Anforderungsspezifikation durchgeführt.

8.4 Leistungstest

Die Leistungstest sollen die Leistungsfähigkeit des Systems Testen.

8.4.1 Härtetest

Der Härtetest soll die Auswirkung einer hohen Auslastung auf das System prüfen. Viele Anfragen auf den Server simulieren genau diesen Fall.

HT 0.0 Mehrfaches gleichzeitiges herunterladen von Kategorien
Testobjekte: -client.logic -client.data -client.communication -server.communication -server.logic -server.data -common.logic **Eingabespezifikationen:** Anzahl der Aufrufe **Ausgabespezifikationen:** Erfolgreiches herunterladen aller Kategorien **Umgebungserfordernisse:** Verbindung zum Internet, Server Online. **Anforderungen:** Mittels einer Main Klasse werden per Multi-Threading n-Fach Kategorien vom Server geladen. **Abhängigkeiten:** IT 0.0, IT 0.1, IT 1.0 IT 1.1

8.4.2 Volumentest

Im Volumentest soll geprüft werden, wie das System auf große Datenmengen reagiert. Simuliert wird dies durch das sequenzielle Anlegen von Begriffen mit maximaler Definitionslänge.

VT 0.0 Sequenzielles hinzufügen von Begriffen
Testobjekte: -client.logic -client.data -client.communication -server.communication -server.logic -server.data -common.logic **Eingabespezifikationen:** Anzahl der einzutragenden Begriffe **Ausgabespezifikationen:** Erfolgreiches herunterladen aller Kategorien **Umgebungserfordernisse:** Verbindung zum Internet, Server Online. **Anforderungen:** Mittels einer Main Klasse werden sequenziell Begriffe auf dem Server bzw. Handy hinzugefügt. **Abhängigkeiten:** IT 0.0, IT 0.1, IT 1.0 IT 1.1

8.4.3 Sicherheitstest

Durch die Verwendung von JBoss wird die Sicherheit des Systems gewährleistet, weil nur auf definiertet Anfragen geantwortet wird. Da es kein Benutzersystem gibt und alle Lerninhalte jedem zur freien Verfügung stehen besteht hier kein Sicherheitsrisiko.

8.4.4 Erholungstest

Beim Erholungstest soll überprüft werden, wie das System auf Fehler reagiert. Eine Simulation ist hier der Abbruch der Verbindung zum Server.

ET 0.0 Verbindungsabbruch
Testobjekte: -client.logic -client.data -client.communication -server.communication -server.logic -server.data -common.logic **Eingabespezifikationen:** keine **Ausgabespezifikationen:** Abbruch des Vorgangs, App läuft wie gewohnt Weiter **Umgebungserfordernisse:** Verbindung zum Internet, Server Online. **Anforderungen:** App Kommuniziert mit Server **Abhängigkeiten:** IT 0.0, IT 0.1, IT 1.0 IT 1.1

ET 0.1 Server Offline
Testobjekte: -client.logic -client.data -client.communication -common.logic **Eingabespezifikationen:** Jegliche Anfrage an den Server **Ausgabespezifikationen:** Fehlermeldung: Server Offline **Umgebungserfordernisse:** Verbindung zum Internet, Server Offline. **Anforderungen:** Benutzer befindet sich in der App **Abhängigkeiten:** IT 0.0, IT 0.1, IT 1.0 IT 1.1